SI-PRÁCTICA 2

Grupo 1 - Prácticas

Lawrence Arthur Rider García

********_

Larg1@alu.ua.es

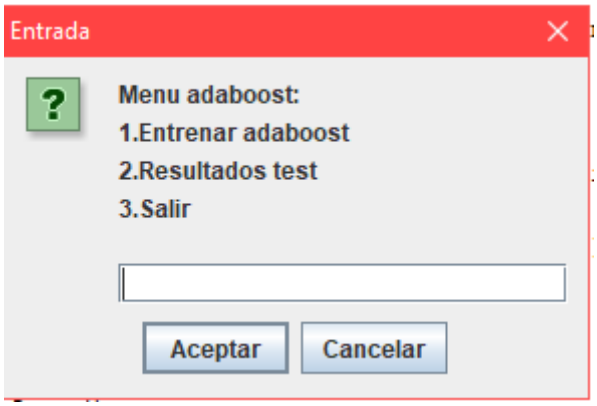
1. Introducción

Para esta segunda práctica hemos vuelto a utilizar el entorno de desarrollo 'NetBeans'.
La práctica consiste en programar un sistema capaz de distinguir entre dígitos manuscritos. Éste será un sistema de aprendizaje automático supervisado. Para ello hemos hecho uso de la base de datos MNIST donde están las imágenes que vamos a utilizar tanto para la parte de aprendizaje como de test.

2. Algoritmo Adaboost

2.1 Explicación del código

En el enunciado pedía argumentos, pero creo que es más intuitivo con una pequeña interfaz donde el usuario es el que proporciona esos valores necesarios. Así que al ejecutarlo aparece un menú con 2 opciones, entrenar o pasar tests. Entrenar guarda en un fichero el resultado, y pasar tests busca ese fichero y se lo aplica a las imágenes de test.



Del proyecto que se nos ha proporcionado para realizar esta práctica he modificado algunas clases. La clase 'Imagen.java' no la he modificado. La clase 'MNISTLoader.java' la he modificado para que funcione con ArrayList en vez de Array y he añadido dos funciones, una devuelve las imágenes de entrenamiento en base al porcentaje que se le pasa y la otra las de test.
He añadido dos clases nuevas, 'ClasificadorDebil' y 'ClasificadorFuerte'.

En 'ClasificadorDebil' hay un constructor de copia para los atributos 'pixel', 'umbral' y 'dirección'. También hay una función estática que genera un ClasificadorDebil al azar y lo devuelve. Las funciones 'aplicarClasificadorDebil' lo que hacen es devolver true o false (en base a la imagen/imágenes que reciben) según el valor que tienen los atributos del clasificadorDebil. Por último, lo que hace el método 'obtenerErrorClasificador' lo que hace es devolver true o false si en realidad lo que está decidiendo el clasificador al aplicarlo es correcto o no.

En ‘ClasificadorFuerte’ hay una función set que guarda los atributos estáticos que se usan luego en las funciones. En la función ‘adaBoost’ recalcar que simplemente he seguido la fórmula, y que hay una condición en la que si cuando se genera un débil al azar éste no tiene un error menor a 0.5 se descarta y se vuelve a intentar y el contador de iteraciones sigue como estaba, esto implica que, aunque solo haya una iteración por cada débil que vaya a formar parte del fuerte, siempre se encontrará uno. Esto lo hice así porque probando el código, si en le ponía que solo probara con una iteración podía darse el caso que encontrara uno con error mayor a 0.5, y al descartarlo por no ser un débil, el fuerte no podía ser generado. También hay otra condición para evitar el sobreentrenamiento donde se comprueba que si el error es 0 que pare de seguir generando débiles.

En la clase ‘Adaboost.java’ se encuentra el main y las diferentes funciones que he ido implementando para su funcionamiento. El main es el que se encarga de mostrar los diferentes diálogos para obtener los parámetros necesarios.

La función ‘entrenarAdaboost’ es la que genera los 10 clasificadores fuertes utilizando la clase ClasificadorFuerte, los guarda en un fichero y muestra los aciertos de estos.

La función ‘pasarTests’ lee de un fichero los 10 clasificadores fuertes y los aplica al % de imágenes que se hayan elegido, al terminar devuelve el resultado por pantalla.

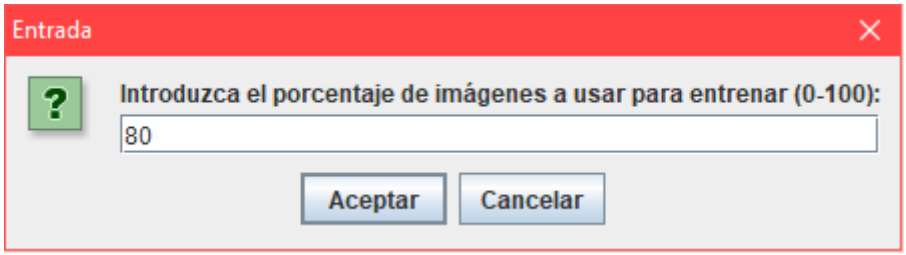
La función ‘getPesos’ devuelve una distribución de pesos, todos con el mismo valor.

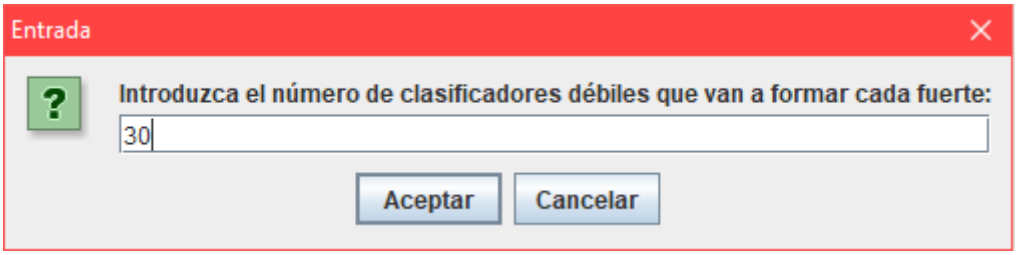
La función ‘getEsperados’ devuelve un arraylist con lo que en realidad es cada imagen, para cada digito los esperados tendrán true en su rango de imágenes.

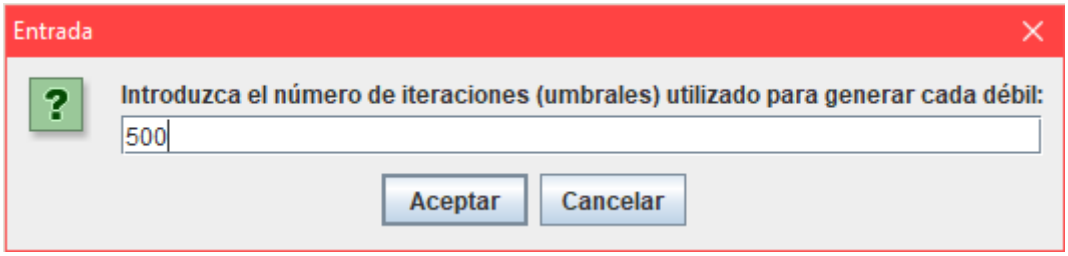
Por último, aclarar que para guardar y leer estos ficheros he usado el formato JSON, para esto he hecho uso de una librería externa (json-simple-1.1.1.jar). He usado esto porque es una tecnología en alza y bastante usada hoy en día, y me parecía interesante y simple de utilizar.

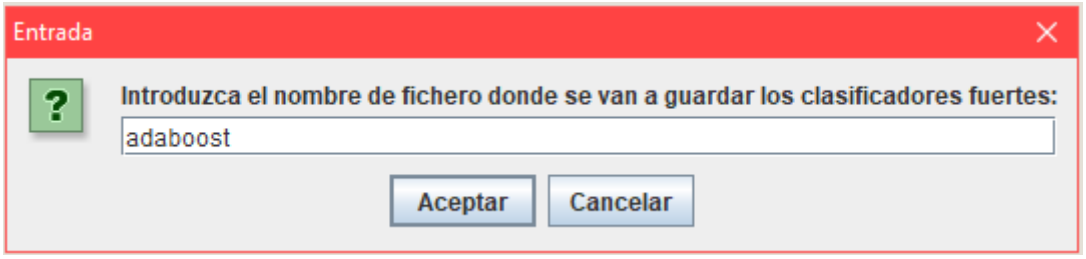
2.2 Ejemplo de ejecución del programa

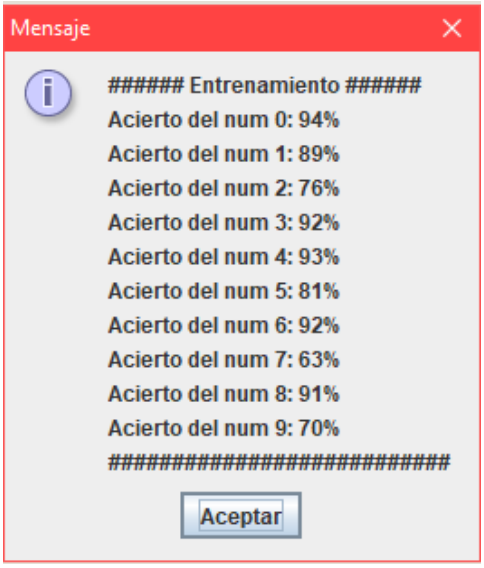
Opción 1:



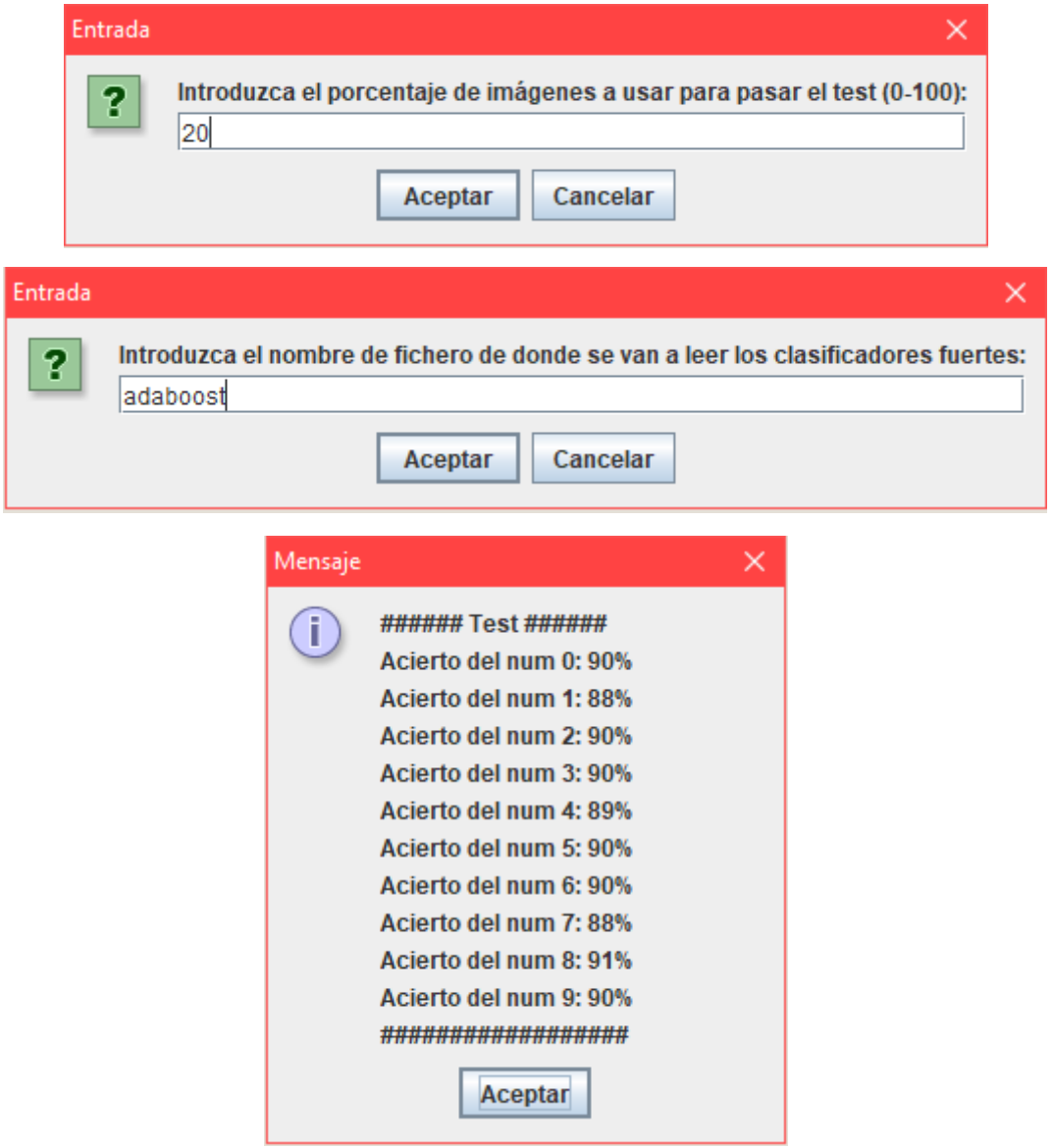








Opción 2:



2.3 Problemas encontrados

La mayoría de los problemas que he encontrado realizando esta práctica han sido en mayor parte sobre entender el algoritmo, y en prácticas han sido solucionados. Por ejemplo, a la hora de calcular el error al principio tenía un contador. Esto implicaba que los pesos no se tenían en cuenta para nada, y por lo tanto funcionara mal. Gracias a esto me di cuenta de que como usamos una gran cantidad de imágenes, a la hora de encontrar un clasificadorDebil siempre se encontraba uno muy bueno pero que en realidad no lo era tanto, y este era el que negaba todas las imágenes. Al haber un gran porcentaje de imágenes de los demás dígitos, aunque negara el dígito que se estaba buscando estaba acertando mucho negando todas las demás. Esto también me pasa ahora y por eso pienso que se podría mejorar el algoritmo, por ejemplo, entrenando con un porcentaje más alto del dígito a entrenar evitando así este problema.

También hay un pequeño problema y es que en la primera/segunda iteración cuando los pesos aun no se han ido modificando, el error calculado no es del todo exacto. Ya que al principio todas las imágenes son igual de probables. Una posible solución que pensé sería ignorar las 5 primeras ejecuciones, por ejemplo, aunque al final no lo implementé porque vi que en general, a pesar de ser un algoritmo bastante básico, funciona bastante bien.

3. Preguntas

¿Cuál es el número de clasificadores que se han de generar para que un clasificador débil funcione?

Con un clasificador se consiguen estos resultados:

```
###### Entrenamiento ######
Acierto del num 0: 9%
Acierto del num 1: 88%
Acierto del num 2: 59%
Acierto del num 3: 90%
Acierto del num 4: 89%
Acierto del num 5: 90%
Acierto del num 6: 44%
Acierto del num 7: 86%
Acierto del num 8: 8%
Acierto del num 9: 72%
############################
```

Como podemos ver, el peor resultado es un 8% lo que implica que se falla el 92% de las veces, un resultado bastante pésimo.

Con 5 clasificadores se consigue el siguiente resultado:

```
###### Entrenamiento ######
Acierto del num 0: 89%
Acierto del num 1: 49%
Acierto del num 2: 70%
Acierto del num 3: 72%
Acierto del num 4: 88%
Acierto del num 5: 90%
Acierto del num 6: 65%
Acierto del num 7: 13%
Acierto del num 8: 91%
Acierto del num 9: 89%
############################
```

Se puede observar que el algunos ya empiezan a acertar más, pero sigue habiendo otros como el 7 que da un 13% de aciertos solo o el 1 con un 49%.

Con 10 clasificadores se obtiene:

```
###### Entrenamiento ######
Acierto del num 0: 93%
Acierto del num 1: 88%
Acierto del num 2: 54%
Acierto del num 3: 82%
Acierto del num 4: 85%
Acierto del num 5: 90%
Acierto del num 6: 90%
Acierto del num 7: 88%
Acierto del num 8: 21%
Acierto del num 9: 89%
############################
```

El acierto más bajo es de 21%, aun sigue siendo insuficiente.

A partir de 25 clasificadores ya se empiezan a obtener aciertos de más del 50%:

```
###### Entrenamiento ######
Acierto del num 0: 90%
Acierto del num 1: 88%
Acierto del num 2: 71%
Acierto del num 3: 91%
Acierto del num 4: 79%
Acierto del num 5: 90%
Acierto del num 6: 90%
Acierto del num 7: 57%
Acierto del num 8: 68%
Acierto del num 9: 61%
############################
```

*Aclarar que solo he puesto 1 iteración para cada débil, si ponemos más cantidad, por ejemplo 500, ya se obtienen mejores resultados.

Con 50 débiles formando cada fuerte, y con 1000 umbrales para cada formar cada débil se consigue este resultado:

```
Acierto del num 0: 90%
Acierto del num 1: 88%
Acierto del num 2: 90%
Acierto del num 3: 90%
Acierto del num 4: 89%
Acierto del num 5: 90%
Acierto del num 6: 90%
Acierto del num 7: 88%
Acierto del num 8: 91%
Acierto del num 9: 90%
```

¿Cómo afecta el número de clasificadores generados al tiempo empleado para el proceso de aprendizaje? ¿Qué importancia le darías?

Cuanto más clasificadores más tiempo tarda, obviamente, lo que si que es verdad que llega un momento que es innecesario añadir más puesto que ya no va a conseguir más porcentaje de acierto o por el sobreentrenamiento, que incluso podría ser negativo.

¿Cómo has dividido los datos en conjunto de entrenamiento y test? ¿Para qué es útil hacer esta división?

He usado un 80-20 pero también es cierto que un 70-30 también funciona correctamente. Es útil para comprobar que está funcionando bien el reconocimiento, ya que si usas las mismas imágenes con las que entrenas para ver si funciona podrías no estar obteniendo resultados reales.

4. Conclusiones

Como conclusión quiero recalcar que he podido observar que es un algoritmo bastante potente y simple, obtiene muy buenos resultados basándose en algo tan simple como varios pixeles y unos valores obtenidos al azar. Hay que decir también que quizás hay varios problemas (ya mencionados antes, como el que niega todas las imágenes) que podrían ser mejorados, pero la verdad que aún con esto es bastante bueno.

En cuanto a lo que he aprendido yo, quiero puntualizar que esta asignatura (más en concreto la práctica) aun no siendo de la rama a la que me quiero dedicar me ha despertado un interés que en principio no tenía, y me ha abierto la mente en ese aspecto. Quizás si hubiera tenido más tiempo me habría gustado poder hacer la parte opcional, pero obviamente es ya culpa mía.

5. Bibliografía

https://www.mkyong.com/java/json-simple-example-read-and-write-json/

PDF de la práctica